# Beadandó feladat dokumentáció

## Készítette:

Szabó-Thalmeiner Bence

Email: [iwb0tm@inf.elte.hu](mailto:iwb0tm@inf.elte.hu)

## Feladat

### Tetris

Készítsünk programot a közismert Tetris játékra. Adott egy 𝑛 × 𝑚 pontból álló tábla, amely kezdetben üres. A tábla tetejéről egymás után új, 4 kockából álló építőelemek hullanak, amelyek különböző formájúak lehetnek (kocka, egyenes, L alak, tető, rombusz). Az elemek rögzített sebességgel esnek lefelé, és az első, nem telített helyen megállnak. Amennyiben egy sor teljesen megtelik, az eltűnik a játékmezőről, és minden felette lévő kocka eggyel lejjebb esik. A játékosnak lehetősége van az alakzatokat balra, jobbra mozgatni, valamint forgatni óramutató járásával megegyező irányba, így befolyásolhatja azok mozgását. A játék addig tart, amíg a kockák nem érik el a tábla tetejét. A program biztosítson lehetőséget új játék kezdésére a pályaméret megadásával (4 × 16, 8 × 16, 12 × 16), valamint játék szüneteltetésére (ekkor nem telik az idő, és nem mozognak az elemek). Ismerje fel, ha vége a játéknak, és jelenítse meg, mennyi volt a játékidő. Ezen felül szüneteltetés alatt legyen lehetőség a játék elmentésére, valamint betöltésére.

### Elemzés

* A játékot 3 pályamérettel kell játszhatóvá tennünk, melyek közül a játékos új játék kezdete előtt fog választani. Emiatt érdemes olyan pályagenerálási módszerrel dolgozni, amivel helyben létrehozhatjuk a táblánkat. Célszrű lesz egy gomb-táblával dolgozni, hiszen ennek generálása egyszerű, valós időben történhet, a gombok pedig idomulhatnak ahhoz, hogy mekkora helyet kell kitöltsenek. A játék irányítása gesture-ökkel lett megoldva, első sorban húzásokkal, és azok irányával dolgozunk.
* A feladatot .NET MAUI alkalmazásként, elsődlegesen Android és Windows platformokon valósítjuk meg. 2 lappal dolgozunk, vertikális és horizontális tájolásban is használható lesz
  + Az első képernyő a játékteret mutatja.
  + A második képernyő a beállításokat adja meg.
* À játék alsó részén információkat találunk a pontokról és az időről, valamint itt található a szünet gomb is. Felül találhatóak az új játék és beállítások gombok, amikkel a játék újraindítását, a mentést/betöltést és a tábla méretét adjuk meg.
* A játékot egy adott méretű nyomógombrács reprezentálja, melynek háttere a tetris tábla állása alapján változik.
* Játék végekor autómatikusan feldob a program egy előugró üzenetet, amiben kiírja az eltelt időt és a pontszámot
* Felhasználói esetek:

Diagram, schematic

Description automatically generated

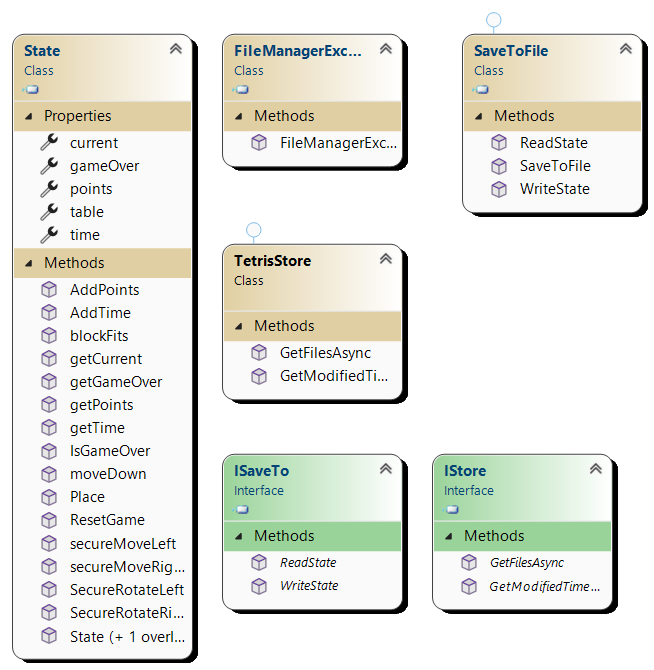
## Tervezés

* Programszerkezet:
* A programot MVVM architektúrában valósítjuk meg, ennek megfelelően View, Model, ViewModel és Persistence mappákban valósítjuk meg az alkalmazáson belül. A program környezetét az alkalmazás osztály (App) végzi, amely példányosítja a modellt, a nézetmodell és a nézetet, biztosítja a kommunikációt, valamint felügyeli az adatkezelést.

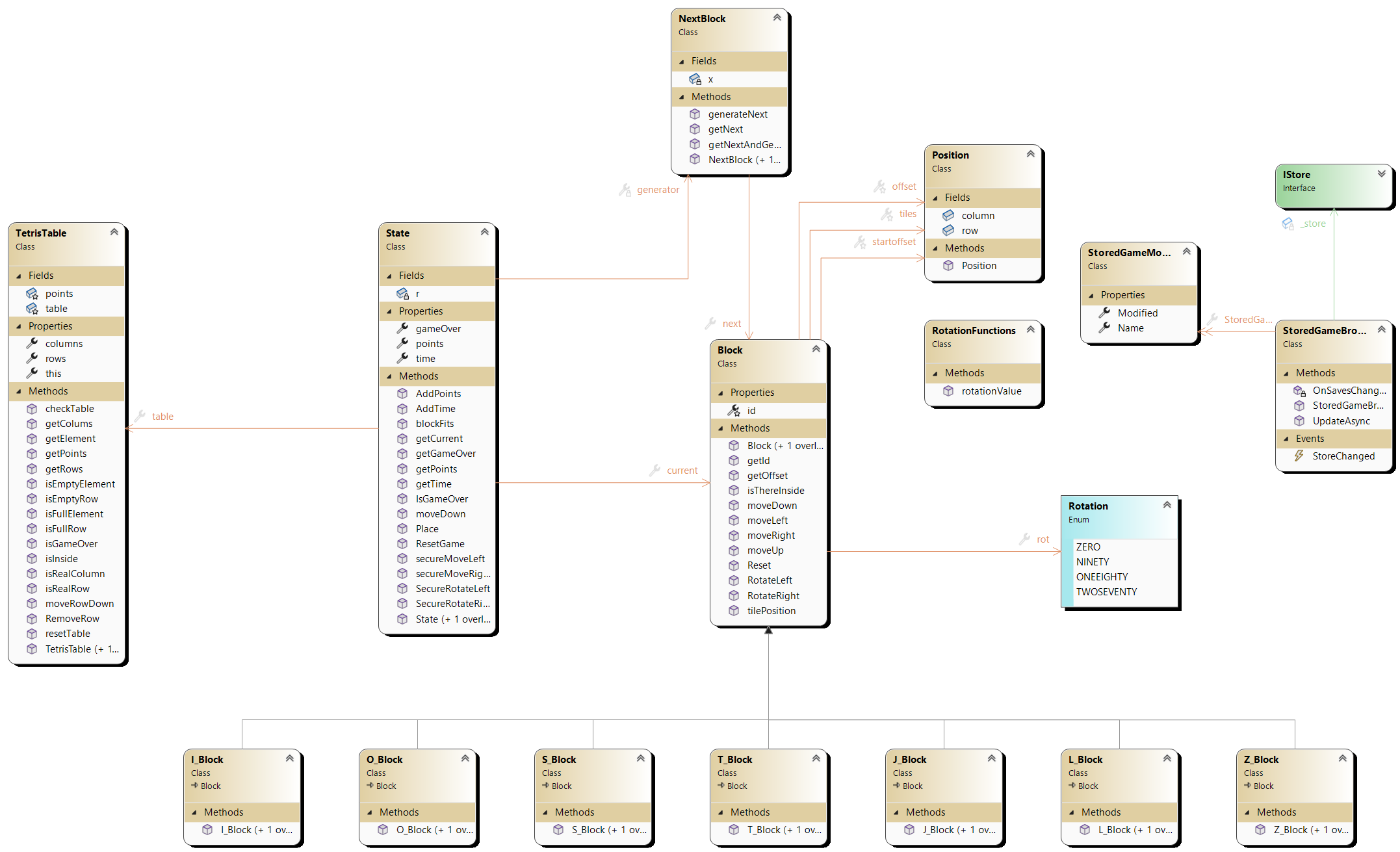
Diagram

Description automatically generated

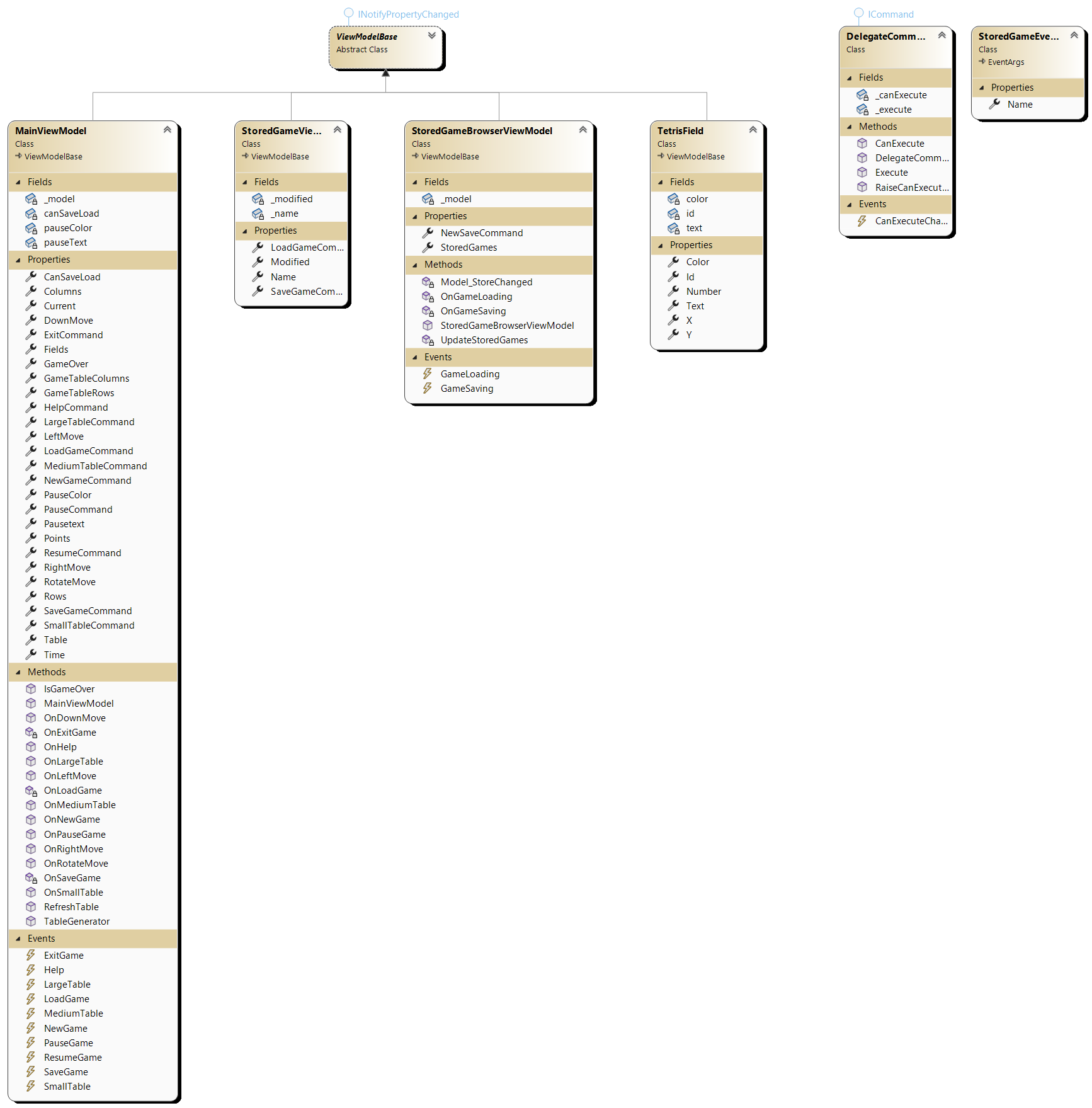
* Perzisztencia:
  + A tetris tábla méreteit, helyzetét és a játék állását jellemző adatokat mentjük egy adott .txt fileba (pontok, idő, és, hogy vége van-e). Bár így csak egy mentésünk lehet, a Tetris-ben afféle checkpoint rendszert megfelelőbb, hiszen a játékos arra használná csak, hogy hibáit semmissé tegye, erre pedig ez az egyszerű megoldás tökéletes, hiszen egy gombnyomásra gyors és hatékony.
  + Hibákat a FileManagerException nevű exceptionnel jelezzük.
  + A mentési rendszert az ISaveTo interface-el fogjuk össze.
  + A SaveToFile class .txt fileba menti a játékállást, vagy a save.txt fileból visszahozza azt.
  + Mivel csak szüneteltetés esetén lehet menteni, így a mentési rendszer szinkron módon ment, nincs szükség háttérben csinálni.



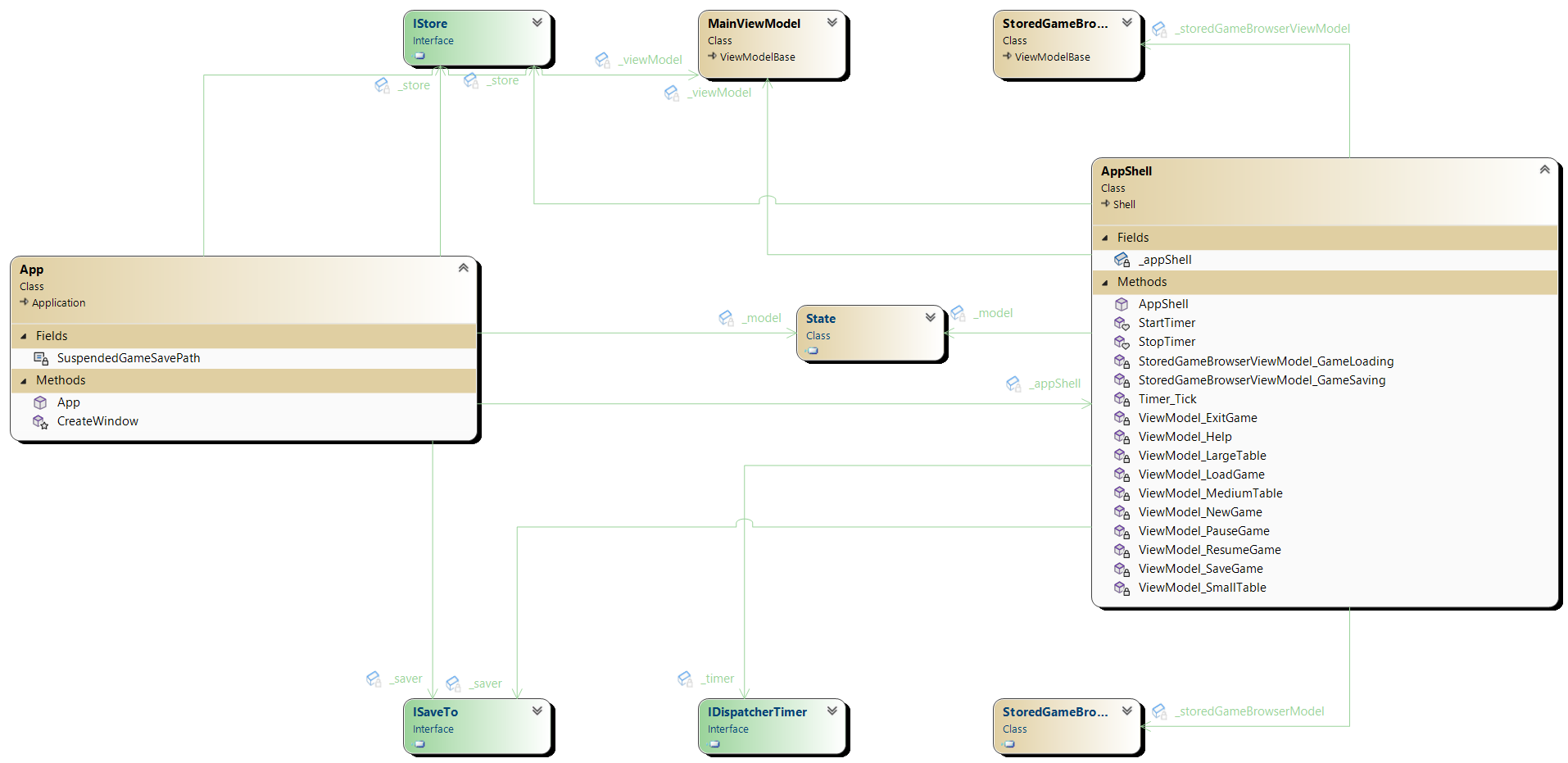
* Modell:
  + A modell lényegi részét a State osztály valósítja meg: Ez tárolja az elért pontszámot és időt (points,time) táblát(table), az új blokkot generáló osztályt (generator) a jelen blokkot (current), valamint azt, hogy vége van-e a játéknak (gameOver). Konstruktor hívásával, valamint a ResetGame metódussal teljesen új játékot kezdhetünk, ebben találjuk a blokk mozgatásához szükséges metódusokat, a lerakás metódusát (Place), valamint gettereket az adattagokhoz.
  + A játékos képes a blokkot mozgatni esés közben metódusok segítségével (moveDown, secureMoveRight,secureMoveLeft, SecureRotateRight )
    - Ezek biztonságos műveletek, ellenőrzik, hogy megteheti-e a játékos a kívánt műveletet.
  + Ha alulról akadályba ütközünk, a Place() metódus behelyezi a blokkot ütközés előtti helyzetében a táblába, majd új blokkot hozunk létre.
  + A blokk osztály tárolja a blokk kinézetét, és, hogy milyen messze van a tábla bal felső sarkától. A blokk addig nem kerül be, amíg meg nem akad, így nem kell a táblát újraírni minden egyes lépésnél.
  + A Blokk alosztályai a blokkok specifikus eseteit tartalmazzák, mind a 4 forgatási lehetőségükkel, valamint a forgatás int-é alakítási metódusával.
  + A Rotation felsorolás a jelen blokk helyzetét határozza meg.
  + A Position osztály 2 számmal megadja, hogy a táblán épp melyik helyzetben kellene legyen az adott elem. Ez az esésénél van felhasználva.
  + A nextBlokk osztály felelős a következő blokk meghatározásáért, lerakás után kap fontos szerepet.



* Nézetmodell
  + A nézetmodell megvalósításához felhasználunk egy általános utasítás (DelegateCommand), valamint egy ős változásjelző (ViewModelBase) osztályt.
  + A nézetmodell feladatait a MainViewModel osztály látja el, amely parancsokat biztosít az új játék kezdéséhez, játék betöltéséhez, mentéséhez,szüneteltetéséhez, folytatásához, a tábla méretének változtatásához, segíység-kéréshez, valamint az elem mozgatásához. A parancsokhoz eseményeket kötünk, amelyek a parancs lefutását jelzik a vezérlőnek. A nézetmodell tárolja a modell egy hivatkozását (\_model), de csupán információkat kér le tőle, illetve a játéknehézséget szabályozza. Direkt nem avatkozik a játék futtatásába, emellett pedig property és változó szinten jelen vannak a kötöttséggel megírt kinézeti elemekhez tartozó információk.
  + A játékmező számára egy külön mezőt biztosítunk (TetrisField), amely eltárolja a pozíciót, szöveget(de csak teszt jelleggel), a gomb színét, valamint egy Number propertyt, ami alapján azonosítani lehet a gombot. A mezőket egy felügyelt gyűjteménybe helyezzük a nézetmodellbe (Fields).
  + A tárolt játékállapotok egy-egy StoredGameViewModel példánnyal írhatóak le. Ezek kollekcióját nem ágyazzuk be a fő nézetmodellbe (SudokuViewModel), hanem a betöltéskor és mentéskor dinamikusan állítjuk elő és adjuk át a nézet számára, bár ennek implementációja elég prototípus-szerű, nem használjuk ki teljesen ennek funkcióit, célja a továbbfejlesztés megkönnyítése.



* Nézet
  + A nézetet lapok segítségével építjük fel.
  + A MainPage osztály tartalmazza a játék fő elemeit: a játéktáblát (dinamikus gombrács), a szünet gombot, az új játék gombot, és a beállítások gombot, amivel a SettingsPage oldalra navigálhatunk.
  + A SettingsPage tartalmazza a mentés és a betöltés gombjait, valamint a kicsi, közepes és nagy táblát összerakó gombokat. Érdemes megjegyezni, hogy a tábla felépítésének hossza eszközfüggő, az alkalmazás nem haladhat tovább a művelet elvégzéséig, hiszen egy félkész tábla manipulálása katasztrofális lenne a program számára.
* Vezérlés
  + Az App osztály feladata az alkalmazás vezérlése, a rétegek példányosítása és az események feldolgozása.
  + A CreateWindow metódus felüldefiniálásával kezeljük az alkalmazás életciklusát a megfelelő eseményekre történő feliratkozással. Így az alkalmazás felfüggesztéskor (Stopped) elmentjük az aktuális játékállást (SuspendedGame), míg folytatáskor vagy újraindításkor (Activated) pedig folytatjuk, amennyiben történt mentés.
  + Az alkalmazás lapjait egy AppShell keretben helyezzük el. Ez az osztály felelős a lapok közötti navigációk megvalósításáért.



## Tesztelés

* A modell funkcionalitása egységtesztek segítségével lett ellenőrizve a TetrisTest osztályban
* Az alábbi tesztesetek kerültek megvalósításra:
  + StateMaker: Státusz alapbeállításainak ellenőrzése, valamint sortelítettség ellenőrző tesztje, ami magában foglalja a helyek telítettségének ellenőrzését is
  + ColumnTest, RowTest: Sorok és oszlopok létezésének ellenőrzése, ezek magukba foglalják a helyek ellenőrzését is
  + SaveAndLoadTest: Perzisztencia teljeskörű vizsgálata: Mentés, betöltés és az eredeti, valamint a visszatöltött státusz elemzése
  + GeneratorTest: A következő blokk generálásának ellenőrzése
  + RotationTest: A rotation elem teljeskörű tesztje
  + TimeTest: Az idő múlásának ellenőrzése